Probabilistic latency for partial ordering

Pascal Urso, Jordi Martori

Probabilistic Latency for Partial Ordering

Pascal Urso et Jordi Martori
Université de Lorraine – Inria
Loria, Nancy, France

Abstract

Ordering events in a distributed systems consists fundamentally in delaying event delivery. Partial ordering, such as FIFO and causal order, has many practical usages in distributed systems and can be obtained in arbitrarily large and dynamic networks. However, partial orderings imply that messages cannot be sent and delivered as soon as produced. In this paper, we examine the latency induced by such partial orderings. We obtain a probabilistic measure of the moment a message can be delivered according the different characteristics of the distributed system. Having such a measure helps to understand the systems behavior and to design new protocols. For instance, our measure allows us to parametrize a naive, albeit efficient, fault-tolerant causal delivery mechanism. We experimentally validate our approach using Internet-scale production distribution latency including faults.

I. Introduction

L'ordonnancement des évènements est un aspect important des systèmes distribués. Une définition traditionnelle d'un système distribué asynchrone est une collection de noeuds où des évènements se produisent. Un évènement est produit par un noeud donné et peut se reproduire ultérieurement sur un autre noeud. Afin de respecter l'intégrité de l'interaction entre les noeuds, un ordre peut être requis entre les évènements. Suivant les contraintes et les besoins du système, cet ordre peut être total, partiel, ou en file (FIFO). Un ordre partiel massivement utilisé est l'ordre causal, tel que défini par la relation "arrivé-avant" [14]. L'ordre causal capture la propriété qu'un évènement qui ne s'est pas produit sur un noeud avant un autre ne peut pas être une cause potentielle de cet autre événement. Donc, dans un ordre causal, un évènement est considéré comme dépendant de tous les évènements qui se sont déroulés avant lui sur le noeud où cet évènement s'est produit en premier.

En raison du théorème CAP [9] – consistance, disponibilité, tolérance aux pannes – et des besoins de passage à l'échelle des systèmes distribués modernes, les ordres partiels ont regagné en intérêt. En effet, le théorème CAP indique que la linéarisabilité, et donc un ordre total entre les évènements, ne peut pas être obtenue dans un système distribué qui requiert une bonne disponibilité et qui peut être sujet aux partitions. A l'inverse, plusieurs solutions récentes visent à assurer une consistance causale pour des systèmes de très grande taille [2], [17]. D'autres utilisent l'ordre causal pour assurer la consistance de types de données distribués tels que les CRDT [18], [22], OT [23] ou [5].

Différents mécanismes existent pour assurer l'ordre FIFO (e.g. les horloges logiques) ou l'ordre causal (e.g. les horloges vectorielles [19] ou les barrières causales [11]). Quoi qu'il en soit, assurer un ordre partiel sur l'événement consiste essentiellement à retarder la livraison de l'événement. Un noeud est prêt à *reproduire* un événement A si et seulement si il a (re)produit tous les événements ordonnés avant A. En supposant une communication arbitraire entre les noeuds, un noeud peut communiquer un événement à un autre noeud qui n'est pas prêt à le reproduire. Dans ce cas, le noeud receveur doit attendre pour obtenir les événements manquants. Nous considérons que la différence entre le temps de réception réel d'un événement et sa reproduction correcte constitue le surcoût en latence induit par l'ordre entre les évènements. Pour chaque système exigeant ou assurant un ordre partiel, il est important de comprendre le coût total en latence.

Cependant, la compréhension de l'impact d'un ordre partiel donné sur la latence n'est pas trivial. La latence réseau de base influe sur la latence totale, mais d'autres facteurs tels que le nombre de noeuds ou le taux de production d'évènements influent également sur celle-ci.

Dans cet article nous étudions comment le respect de différents ordres partiels impactent la latence des systèmes distribués. Nous exprimons d'une manière probabiliste de telles latences. Nous motivons le besoin cette étude, notamment en définissant un mécanisme de délivrance fiable. Aussi, nous conduisons une validation expérimentale en utilisant des distributions de latences et de fautes qui proviennent d'entreprises de taille mondiale.

II. Motivation

Nous nous inspirons du travail sur "l'obsolescence bornée probabiliste" (PBS) [3]. Les auteurs proposent de fournir des prévisions probabilistes sur le comportement de la consistance inéluctable dans les systèmes à quorum partiels. Comme potentiels travaux futurs, ils envisagent d'étudier des consistances plus fortes telle que la consistance causale. Cependant, ils indiquent que les bornes probabilistes dans ce cas serait très élevées. Nous sommes d'accord avec ce fait dans le cadre d'un modèle simplifié du système. En effet, dans un système distribué des fautes ou des congestions ont lieu et des messages peuvent être perdus ou extrêmement retardés. Les messages qui dépendent de messages perdus ne vont plus pouvoir être reproduits. Et rapidement, tout le système se retrouvera bloqué.

Cependant, des systèmes distribués de grande taille, servant des milliards d'utilisateurs réels, utilisent des ordres partiels [26], voir même des ordres totaux [6]. Ces systèmes ont besoin, justement à cause des bornes théoriques élevées, d'une connaissance fine du comportement de la latence.

Par exemple, la fiabilité de la livraison des messages est souvent assurée par des mécanismes de confirmation. Cependant, les messages de confirmation ont un coût élevé

en charge réseau [16]. Dans le cas d'une diffusion totale – chaque message doit être confirmé par chaque autre noeud – les messages de confirmation représentent donc potentiellement plus de $N-1$ fois la charge initiale [1]. A la place, nous proposons de détecter les problèmes à l'arrivée des messages en utilisant les informations d'ordre entre les messages. En effet, si un message reçu ne peut pas être reproduit car un autre message est manquant, le noeud receveur peut demander le message manquant. Bien sur, il se peut que le message manquant soit seulement un peu retardé et arrive alors deux fois si il est redemandé trop tôt. Il est donc important de comprendre finement la latence du système pour paramétrer ce mécanisme afin de maîtriser le nombre de message inutiles. Dans la section IV-E nous détaillons ce mécanisme et dans la section V-B nous évaluons ses performances.

Comprendre la latence due à la causalité est aussi utile dans des contextes applicatifs. Par exemple, dans le travail collaboratif, où le respect de la causalité est important [24], un mécanisme de conscience de groupe [7] est extrêmement utile. Ce mécanisme permet notamment de savoir comment les autres participants sont en train de travailler, de manière parfois détaillée et en temps réel [15]. Une bonne connaissance de la latence, permettrait d'avoir aussi une information sur la diffusion du travail de l'utilisateur.

Un dernier exemple concerne les types de données répliqués [5], [22]. Ces types de données permettent de définir des structures de données partagées pouvant être modifiées sans verrou et de manière pair-à-pair. Certain de ces algorithmes ont besoin d'un ordre causal pour assurer leur consistance, mais d'autres n'en ont pas besoin [20]. Par exemple, si nous prenons l'algorithme logoot [27], celui-ci a besoin de la causalité uniquement pour empêcher de la suppression d'un élément avant sa création (et donc sa non-suppression sur certain noeuds). Avec une connaissance de latence, on pourrait remplacer le maintien de l'ordre causal, qui est coûteux, par une simulation de celui-ci et assurer la consistance par le renvoi périodique – et paramétré grâce à cette connaissance – des opérations de suppression.

III. Définitions

Dans cette section, nous donnons quelques définitions pour les concepts théoriques clés sur lesquels nous travaillons. Premièrement, nous définissons un système distribué comme un ensemble non borné de noeuds communicants sur lesquels des évènements se produisent. Chaque noeud possède sa propre mémoire, sa propre horloge, et communique avec les autres par l'envoi de messages. Théoriquement, nous considérons que la communication est fiable – i.e. que chaque message envoyé arrivera à destination – mais que la durée d'arrivée de chaque message peut être arbitrairement longue. [2]

Bien que certains de nos résultats puissent s'appliquer à un cadre plus général, nous considérons dans nos travaux qu'un évènement se produit initialement sur un noeud, et qu'il est envoyé sur tous les autres noeuds pour être reproduit. Ce modèle de communication est notamment utilisé dans les systèmes de réplication optimiste où chaque évènement représente une modification des données et doit être appliqué sur toutes les répliques de cette donnée [2], [5], [17], [18], [22], [23].

1. Les messages de confirmation peuvent être groupés, réduisant ainsi la surcharge réseaux, mais entraînant un surcoût en latence.

2. La perte d'un message est représentée par un message mettant un temps irraisonnable – tel que 10^{10} unité de temps – pour arriver.

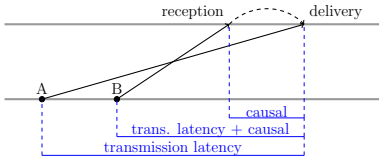

FIGURE 1: Ordre FIFO

A. Ordre FIFO

L'ordre FIFO (ou en file) suppose que tous les évènements qui se sont produits sur un noeud doivent se reproduire dans le même ordre sur tous les noeuds. Il s'agit d'un ordre partiel car deux évènements qui se sont produits sur des noeuds différents peuvent être reproduits dans n'importe quel ordre. Une méthode simple pour assurer un ordre FIFO est d'utiliser une horloge logique par noeud. La figure 1 représente un cas où un deux messages d'un même site se croisent – par exemple en empruntant deux chemins différents dans le réseau – et la livraison de l'évènement B ne peut réellement se produire qu'après la livraison de l'évènement A.

B. Ordre causal

L'ordre causal est un ordre plus strict que l'ordre FIFO. Tout couple d'opération ordonné selon l'ordre FIFO est aussi ordonné selon l'ordre causal.

La causalité est définie comme la relation entre la cause et l'effet. Dans les systèmes distribués, la relation d'ordre causal ($\rightarrow$) [14] nous dit quels événements eu lieu avant la production des autres. L'ordre causal capture toutes les causalités potentielles. En effet, si un évènement A s'est (re)produit sur un noeud avant un autre évènement B, alors il pourrait potentiellement être la cause de B. Un noeud ne devant pas appliquer l'effet avant la cause, il ne faut pas exécuter B avant A. Par exemple, dans un système de fichier distribué, il ne faudrait pas créer un fichier dans un répertoire avant de créer ce répertoire.

La figure 2 présente un cas où le respect de la relation entraîne un délai supplémentaire. L'événement A a été remis sur le noeud 2 avant que B se produise, donc, B dépend de A ($A \rightarrow B$). L'évènement B ne peut donc pas être reproduit avant A, et sur le noeud 1, la reproduction de l'évènement B doit être retardée.

Pour détecter et assurer le respect de l'ordre causal, plusieurs mécanismes existent. Essentiellement, le système doit expliciter la relation d'ordre en utilisant des horloges vectorielles [8], [19] ou des barrières causales [21] – qui représentent les opérations maximales ordonnées avant une opération donnée.

Une horloge vectorielle H a une entrée H_x par noeud x dans le système. Chaque noeud possède une horloge qui est, pour chaque entrée, le maximum des entrées correspondantes de toutes les estampilles des messages qu'il a (re-)produit.

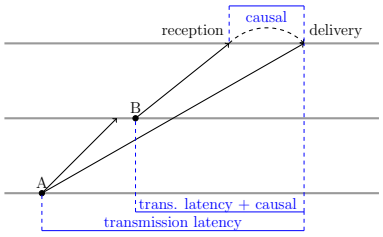

FIGURE 2: Ordre causal

Chaque message A a une estampille H^A qui est la valeur de l'horloge du noeud qui a produit ce message au moment où il l'a produit. On a $A \rightarrow B$ si est seulement si $H^A < H^B$ c'est-à-dire :

$$\forall x.[H_x^A \leqslant H_x^B] \wedge \exists y.[H_y^A < H_y^B]$$

Si ni $H^A < H^B$ ni $H^B < H^A$ alors A et B sont concurrents.

En utilisant ces informations, le système retarde l'application des évènements sur les noeuds qui ne sont pas prêts, i.e. qui n'ont pas reçu tous les évènements dépendants. Ce retard peut être géré à l'émission ou, pour plus d'efficacité, à la réception du message portant l'évènement. Le retard induit correspond au le *coût en latence de la causalité*.

La causalité implique d'autre coûts. Notamment, le coût en bande passante du à l'information supplémentaire permettant de représenter l'ordre causal. Pour N noeuds dans le système, ce coût est de $O(N^2)$ par message, dans le cadre général, et de $O(N)$ dans le cadre qui nous intéresse, la diffusion globale [19].

C. Latence et visibilité

Nous distinguons la latence "réseau" de la latence "système". La latence "réseau" est le temps que met un message pour arriver d'une source à une destination. Cette latence est impactée par les limitation physique, l'utilisation du réseau et son implémentation. C'est à dire par des paramètres souvent extérieurs aux systèmes distribués. La latence "système" est le temps entre la production originale d'un évènement et sa reproduction sur un autre noeud. Cette latence est impactée par la latence réseau, par le protocole utilisé et par son implémentation.

La visibilité, définie dans [3], mesure le temps que met un évènement pour être observable sur tout le système ; autrement dit le maximum des latences pour un message donné.

IV. LATENCE PROBABILISTE POUR LES ORDRES FIFO ET CAUSAL

Dans cette section, nous introduisons la notion de latence probabiliste pour des ordres partiels, qui décrit comment se comporte latence dans les systèmes distribués qui utilisent un ordre partiel et une diffusion totale pour passer des messages aux autres noeuds. On introduit également un système de livraison fiable qui utilise notre modèle pour détecter les des messages problématiques.

Ces latences sont définie dans le cas général en utilisant probabilités sur des variables aléatoires. Puis nous évaluons ces probabilités en supposant que les délais de transmission des messages suivent un modèle de distribution exponentielle.

Afin de définir quand un message pourra être re-produit à la destination, nous avons utilisé différents éléments. L'élément de base est la probabilité qu'un message soit reçu sur un noeud t temps après son envoi. Par reçu, nous entendons une livraison mais pas forcement une re-production, qui elle dépend de l'ordre entre les messages. Cette probabilité est noté $P(X \leqslant t)$, avec X la variable aléatoire représentant le temps que ce message arrive. Pour une distribution exponentielle, cela correspond à la fonction de répartition de la loi exponentielle $P(X \leqslant t) = 1 - e^{-\lambda t}$ avec λ^{-1} la latence espérée dans la communication.

A. Ordre FIFO

Dans le cas d'un ordre FIFO, un message ne peut être reproduit que si tous les messages provenant du même noeud ont déjà était reproduit. Donc la probabilité que le $k^{\text{ième}}$ message X_k d'un noeud puisse être re-produit à un temps t est égale à la probabilité que :

— le message soit arrivé $P(X_k \leqslant t)$, et

— tous les messages précédents de ce noeud soit arrivés $P(X_i \leqslant t + \Delta_i)$ avec Δ_i le temps séparant la production du message i du message k.[3]

Ceci nous donne la probabilité suivante (en posant $\Delta_k = 0$) :

$$\mathbb{P}(F_k \leqslant t) = \prod_{i=1}^{k} \mathbb{P}(X_i \leqslant t + \Delta_i) \tag{1}$$

Et dans le cas exponentiel,

$$\mathbb{P}(F_k \leqslant t) = \prod_{i=1}^{k} (1 - e^{-\lambda(t+\Delta_i)}) \tag{2}$$

Considérons maintenant la visibilité du message k. Comme mentionné précédemment, nous supposons un système de diffusion globale, ce qui signifie que chaque message est envoyé à toutes les répliques en même temps.[4] Nous pouvons généraliser l'équation 1 en se demandant quelle est la probabilité que, après t temps, tous les autres noeuds aient reçu le message. Si nous supposons un système avec N répliques, et que chaque diffusion se fait de manière indépendante, la probabilité est :

$$\mathbb{P}(VF_k \leqslant t) = (\prod_{i=1}^{k} \mathbb{P}(X_i \leqslant t + \Delta_i))^{N-1} \tag{3}$$

B. Ordre causal

Dans l'ordre causal, un message est dépendant des messages produit avant lui mais aussi des message re-produits (provenant d'un autre noeud). Nous établissons la probabilité qu'un message émis par un noeud soit reçu avant t temps après avoir été reçu sur un autre noeud. C'est à dire que la différence

3. Chaque message a eu plus de temps pour arriver.
4. Nous faisons abstraction du temps pris par le mécanisme d'envoi.

entre le temps d'envoi du message entre l'origine et un noeud et le temps d'envoi sur un autre noeud est inférieur à t.

$$\mathbb{P}(R \leqslant t) = \mathbb{P}(X - Y \leqslant t) \tag{4}$$

Pour le cas exponentiel cela nous donne :

$$\mathbb{P}(R \leqslant t) = 1 - 0,5e^{\lambda t} \tag{5}$$

Intuitivement, au moment où un noeud reçoit un message ($t = 0$), il y a une chance sur deux qu'un autre noeud l'ai reçu. Plus formellement :

Proof: Les deux processus X et Y ont la même espérance λ^{-1}, et ont une densité de probabilité de $\lambda e^{-\lambda x}$ et $\lambda e^{-\lambda y}$. Comme ils sont indépendants, nous pouvons exprimer la densité commune $\lambda^2.e^{-\lambda x}.e^{-\lambda y}$. Donc

$$\mathbb{P}(R \leqslant t) = \mathbb{P}(X - Y \leqslant t) = \iint \lambda^2.e^{-\lambda x}.e^{-\lambda y}\, dxdy$$

Pour $Y \leqslant X + t$:

$$\mathbb{P}(R \leqslant t) = \lambda^2 \int_0^\infty e^{-\lambda x}(\int_0^{x+t} e^{-\lambda y}\, dy)\, dx$$

En résolvant l'intégrale interne en $\frac{1-e^{-\lambda(x+t)}}{\lambda}$, nous avons :

$$\mathbb{P}(R \leqslant t) = \lambda \int_0^\infty (e^{-\lambda x} - e^{-\lambda.t}.e^{-2\lambda.x})\, dx$$

Ce qui se résout en

$$\mathbb{P}(R \leqslant t) = \lambda(\frac{1}{\lambda} - \frac{e^{-\lambda.t}}{2\lambda})$$

et donc en $1 - 0,5e^{-\lambda t}$ ■

Maintenant, nous pouvons exprimer la probabilité qu'un message soit causalement prêt. Pour que le message d'un noeud arrive a destination, il faut que

— ce message et tous les messages précédents de ce noeud soit arrivés (voir 1)

— tous les messages reçus (et re-produits) soient arrivés

Donc pour tous les événements i (re-)produit sur ce noeud, avec δ_i la différence en le temps d'émission de k et 1) le temps d'émission de i pour un évènement local 2) le temps de réception de i pour un évènement distant.

$$\mathbb{P}(C_k \leqslant t) = \prod_{i=1}^{k} \begin{cases} \mathbb{P}(X_i \leqslant t + \delta_i) & \text{si le noeud a produit } i \\ \mathbb{P}(R_i \leqslant t + \delta_i) & \text{sinon} \end{cases} \tag{6}$$

Ce qui fait dans le cas exponentiel

$$\mathbb{P}(C_k \leqslant t) = \prod_{i=1}^{k} \begin{cases} 1 - e^{\lambda(t+\delta_i)} & \text{si le noeud a produit } i \\ 1 - 0,5e^{\lambda(t+\delta_i)} & \text{sinon} \end{cases} \tag{7}$$

Pour la visibilité de l'évènement il faut compter $N - 1$ noeuds devant recevoir les évènement locaux et $N - 2$ noeuds devant recevoir les évènements distants.

$$\mathbb{P}(C_k \leqslant t) = \prod_{i=1}^{k} \begin{cases} (P(X_i \leqslant t + \delta_i))^{N-1} & \text{si le noeud a prd. } i \\ (P(R_i \leqslant t + \delta_i))^{N-2} & \text{sinon} \end{cases} \tag{8}$$

C. Optimisation

Les équations définies précédemment utilisent les probabilités de réception de tous les évènements précédents. Hors, plus on considère des évènements anciens, plus l'impact sur la latence de l'évènement considéré devient négligeable. Pour améliorer les performances du calcul, on peut ne considérer que les évènements ayant une probabilité inférieure à un certain quantile p (par exemple 1%) de ne pas être arrivés sur tous les noeuds. Pour cela il faut calculer le temps $t(p)$ tel que :

$$\mathbb{P}(X \leqslant t(p))^{N-1} = 1 - p \tag{9}$$

Pour une loi exponentielle, nous avons

$$t(p) = \frac{ln(1 - \sqrt[n-1]{1-p})}{-\lambda} \tag{10}$$

Par exemple pour $\lambda = 0,5$; il y a plus 99% de chance qu'un évènement envoyé il y a plus de $13,59s$ sont reçu sur 10 noeuds.

Proof: Pour loi de probabilité exponentielle,

$$\mathbb{P}(X \leqslant t(p))^{N-1} = (1 - e^{-\lambda.t_m})^{n-1} = 1 - p$$

et donc comme $n - 1 > 0$

$$1 - e^{-\lambda.t(p)} = \sqrt[n-1]{1-p}$$

puis

$$-\lambda.t(p) = ln(1 - \sqrt[n-1]{1-p})$$

■

Ainsi suivant la précision p voulue, on peut calculer les valeurs de latence probabiliste que pour les opérations i ayant $\delta_i < t(p)$.

D. Généralisation

Les définitions ci-dessus nous donne une définition précise de la latence et de la visibilité attendue pour un évènement particulier. Cela peut être particulièrement utile pour certain cas applicatif, par exemple pour les mécanismes de conscience de groupe [15], où il faut donner à l'utilisateur des information concernant chaque évènement en particulier. Cependant nous pouvons généraliser ces équations pour obtenir une vision globale d'un système.

En supposant que les évènements se produisent de manière uniforme selon un taux de R – soit un taux de $r = R/N$ pour chaque noeud –, et en utilisant la définition $t(p)$ ci-dessus, nous obtenons une version généralisée de la description de la latence FIFO (cf. équation 1) :

$$\mathbb{P}(F \leqslant t) = \prod_{i=0}^{\lceil t(p).r \rceil} \mathbb{P}(X \leqslant t + \frac{i}{r}) \tag{11}$$

En effet, on considère $t(p).r$ évènements non-négligeables par noeud et deux évènements sont séparés en moyenne d'une fréquence r^{-1}.

Pour la latence causale (cf. équation 8) nous obtenons

$$\mathbb{P}(C \leqslant t) = \begin{array}{l} \prod_{i=0}^{\lceil t(p).r \rceil} \mathbb{P}(X \leqslant t + \frac{i}{r}). \\ \prod_{i=1}^{\lceil t(p).r.(n-1) \rceil} \mathbb{P}(R \leqslant t + \frac{i}{r(n-1)}) \end{array} \quad (12)$$

On considère $t(p).r.(n-1)$ évènements non-négligeables [5] par noeud et une réception uniforme avec une fréquence de $(r(n-1))^{-1}$.

E. Livraison fiable

Nous proposons d'utiliser un mécanisme de livraison fiable très simple. Plutôt que de demander une confirmation à tous les noeuds pour chaque évènement produit et donc envoyé, nous détectons les problèmes potentiels à l'arrivée des messages.

Le principe est d'utiliser le mécanisme de délivrance causale (ou FIFO) déjà nécessaire. En effet, lors de la réception par le réseau d'un message, celui-ci décide si ce message peut être appliqué ou pas. Le message peut être appliqué si tous les messages dépendants on été reçus. Par exemple, en utilisant des horloges vectorielles [19], un message produit par un noeud x et ayant une estampille M peut être re-produit sur un noeud y ayant une horloge H ssi

$$H_x = M_x + 1 \wedge \forall z \neq x.[H_z \geqslant M_z]$$

Donc un message manquant est

- un message numéro $H_x + i$ du noeud x tel que $i > 0$ et $H_x + i < M_x$
- ou un message numéro $H_z + i$ du noeud $z \neq x$ tel que $i > 0$ et $H_z + i \leqslant M_z$

Le noeud y peut demander ce message au noeud z, ou bien au noeud x – celui-ci connaissant ce message car il a envoyé un message dépendant de celui-ci. Dans le cas FIFO, seul un message du noeud x peut être détecté comme manquant.

Ce mécanisme est simple mais requiert pour fonctionner de manière efficace, deux conditions

1) le taux de production des évènements doit être suffisamment élevé pour détecter tôt un problème.
2) ne pas détecter trop de faux positifs.

Si le mécanisme demande un message manquant, et reçoit celui-ci par le mécanisme normal de livraison avant de le recevoir par ce mécanisme, nous appelons ceci un *faux positif*. Si, mais seulement si, nous obtenons trop de faux positifs ($> 50\%$) le mécanisme n'est pas plus efficace en terme de charge réseaux qu'un mécanisme classique de confirmation. Pour diminuer le nombre de faux positif, nous pouvons attendre après réception du message une certaine durée "*waiting time*" avant de considérer un message dépendant comme manquant.

Nous avons alors pour chaque message potentiellement manquant, trois cas (voir figure 3) en considérant le temps d'arrivée "normal" du message :

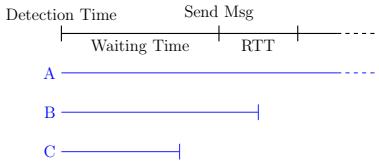

FIGURE 3: Temps d'arrivée et détection

- **A :** Le message est réellement perdu ou arrive après une demande du mécanisme (RTO). Nous avons alors un gain en latence.
- **B :** Le message arrive avant la demande du mécanisme. Nous n'avons pas de perte en latence, mais une perte en charge du réseau (faux positif).
- **C :** Le message arrive pendant le temps d'attente. Nous n'avons ni perte en latence ni charge supplémentaire.

Bien sur plus le temps d'attente sera court, plus il y aura de faux positif, mais aussi une latence causale ou FIFO plus courte. De plus, pour assurer un livraison totalement fiable, si le message n'est toujours pas reçu après un cycle (RTO), il faut le redemander à nouveau, jusqu'à réception, comme dans un mécanisme de confirmation classique. Dans nos expérimentations, nous utilisons un délai de retransmission (RTO) usuel provenant l'algorithme Jacobson/Karels – $RTT + 4D$ – [12].

Notre étude de la latence nous permet d'évaluer le nombre de faux positif. Soit la probabilité qu'un message A produit δ temps avant un message B soit détecté comme un faux positif par B après un temps d'attente t :

$$f(\delta) = \begin{array}{l} (\frac{N-1}{N}\mathbb{P}(C \leqslant \delta) + \frac{1}{N}). \\ \mathbb{P}(Y + t + \delta < X \leqslant Y + Z + Z' + t + \delta) \end{array} \quad (13)$$

En effet, il faut :

- que $A \rightarrow B$, càd. que soit A et B proviennent du même noeud (probabilité $1/N$), soit qu'ils proviennent de deux noeuds différents (probabilité $(N-1)/N$) et que A est été reçu et soit causalement prêt avant l'envoi de B ($\mathbb{P}(C \leqslant \delta)$) [6]
- que A soit reçu au moins t temps après B ($\mathbb{P}(X > \delta + Y + t)$
- que A soit reçu avant le message de réponse à la demande du mécanisme ($\mathbb{P}(X \leqslant \delta + Y + t + Z + Z')$)

L'équation ci-dessus nous permet de calculer l'espérance de faux positif $FP(t)$ pour tout message en posant le message B et en prenant en compte tout message A potentiel en intégrant l'expression ci-dessus $FP(t) = \int_O^\infty f(\delta)\, d\delta$.

Ce qui nous donne dans le cas exponentiel (avec $\mathbb{P}(C \leqslant \delta)$ tel que défini en (12))

$$\int_O^\infty [(\frac{N-1}{N}\mathbb{P}(C \leqslant \delta) + \frac{1}{N}).\frac{3}{2^3}e^{-\lambda.(t+\delta)}]\, d\delta \quad (14)$$

5. Pour des raisons d'efficacité, on pourrait diminuer ce nombre car ces évènements étant distants, ils ont plus de chance d'être reçu sur tous les noeuds et donc d'être négligeable.

6. Comme le mécanisme ci-dessus permet d'améliorer la latence du système, il influence la probabilité $\mathbb{P}(C \leqslant t)$. Cette probabilité sera donc plus élevé et notre calcul doit être un calcul conservatif, donnant une borne supérieur à la probabilité de faux positif.

Nous n'avons pas encore résolu cette intégrale, mais ont peut obtenir une valeur approchée grâce à une intégration numérique par la méthode des rectangles $\sum_{\delta=O}^{\infty} f(\delta)$ car, rapidement, quand δ grandi, la valeur $f(\delta)$ devient négligeable à cause du terme $\frac{1}{2}e^{-\lambda.(t+\delta)}$.

V. Résultats expérimentaux

Nous avons développé un outil en utilisant le langage R [25]. La première étape pour créer une simulation consiste à définir la configuration pour le scénario. Les paramètres d'un scénario sont

- la distribution statistique des latences réseaux représentée par une loi mélange exponentielle – pareto,
- le taux global de production d'évènements,
- le nombre de noeuds,
- la durée de la simulation et la fenêtre d'observation situé au milieu de la simulation,
- le nombre de fois où le scénario sera instancié et analysé pour produire les résultats

La phase de simulation consiste simplement à produire des temps $p(i)$ pour des évènements de manière uniforme sur la durée de la simulation et sur l'ensemble des noeuds. Puis, pour chaque évènement on produit un temps d'arrivée $r(i, y)$ du message pour chaque autre noeud y égal à $p(x)$ plus une valeur aléatoire générée suivant la loi. On pose $r(i, n(i)) = p(i)$ pour le noeud $n(i)$ ayant "produit" l'évènement.

A partir de ces temps d'arrivée, nous calculons le temps où l'évènement peut être re-produit en suivant l'ordre donné : $f(i, x)$ pour FIFO ou $c(i, x)$ pour l'ordre causal. Pour chaque opération i dans l'ordre $p(i)$ croissant, nous calculons ses dépendances $dep(i)$ – i.e. les évènements j tel que pour l'ordre FIFO :

$$f(j, n(i)) < p(i) \wedge n(j) = n(i)$$

et pour l'ordre causal :

$$c(j, n(i)) < p(i)$$

Enfin nous pouvons calculer la latence FIFO et causale pour chaque autre noeud , latences qui sont le maximum de tous les temps de réception :

$$c(i, x) = max_{j \in (dep(i) \cup \{i\})}(r(j, x))$$

Ensuite, nous pouvons calculer la visibilité :

$$vf(i) = max_x(f(i, x)) \text{ et } vc(i) = max_x(c(i, x))$$

A. Coût de causalité

Les figures 4 et 5 nous permettent de comprendre quel est le coût du respect de l'ordre causal. Premièrement, nous étudions la relation entre la latence réseau et la latence causale. Les paramètres de la simulation sont

- une distribution exponentielle de paramètre λ pour les latences réseaux
- un taux de production global de 5
- 10 noeuds
- un simulation de 1000 évènement, et une fenêtre d'observation de 800.

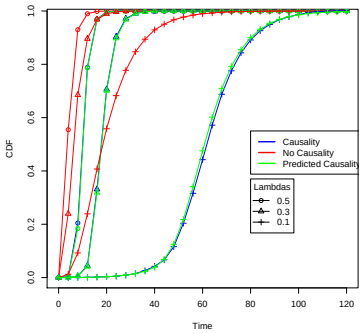

FIGURE 4: Coût en latence de la causalité suivant différentes latences réseau

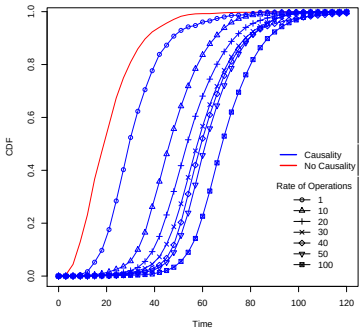

FIGURE 5: Coût en latence de la causalité suivant différents taux de production

- 10 répétitions

Dans la figure 4, nous faisons varier λ sur les valeurs $0,1$, $0,3$ et $0,5$. Pour chaque λ, nous présentons la fréquence cumulée (CDF) de la visibilité des évènements en fonction du temps 1) sans prendre en compte d'ordre en rouge, 2) avec un ordre causal en bleu, 3) calculée en utilisant notre formule probabiliste en vert. Nous notons tout d'abord que notre simulation correspond très fortement à notre prédiction probabiliste. Ensuite, nous voyons que la latence causale semble décalée par rapport à la latence réseau. Elle possède la même forme mais avec un décalage d'autant plus grand que la latence réseaux est élevée.

Dans la figure 5, nous avons $\lambda = 0,1$ et nous faisons varier le taux de production entre 1 et 100. Nous voyons que plus y a d'évènements, plus la latence causale est élevée, alors que la visibilité réseau est – logiquement – non affectée [7]. En effet, plus il y a d'évènements dans le système, plus chaque évènement sera retardé par la causalité car il y a plus de risque d'avoir un évènement manquant. Ce résultat justifie le fait d'avoir une définition propre à l'ordre partiel considéré.

7. Dans un environnement de production réel, un très haut de production influencera la congestion réseau et donc la latence.

Conf. Name	pareto dist. (%)	Scale (X_m)	Shape (α)	lambda (λ)
LNKD-SSD	91.22%	0.235	10	1.66
LNKD-HDD	38%	1.05	1.51	0.183
YMMR-W	93.9%	3	3.35	0.0028
YMMR-R	98.2%	1.5	3.8	0.0217

TABLE I: Paramètres de distribution provenant de systèmes réels en production [3]

B. *Délivrance fiable*

Pour évaluer le mécanisme de délivrance fiable nous utilisons des distributions de latence provenant de systèmes réels en production (table I). Ces distributions proviennent des entreprises LinkedIn (LNKD) et Yammer (YMMR). Ce sont des distribution suivant une loi mixte exponentielle - pareto. Elles représente des distributions réelles à queue longue qui peuvent contenir latences problématiques. Par exemple le 99ème centile de distribution pareto de YMMR-W vaut 76 fois la médiane.

Les autres paramètres de la simulation sont 1) un taux de production $r = 45$ (équivalent au nombre d'écritures[8]) par seconde de [3] 2) 5 noeuds 3) une durée de 10000 évènements et une fenêtre d'observation de 8000 évènements.

Les résultats sont présentés figure 6. Pour chaque distribution, nous présentons en noir la fréquence cumulée (CDF) de la visibilité "réseau" (sans ordre) en fonction du temps. En rouge, vert et respectivement bleu, nous indiquons la visibilité causale avec le mécanisme de livraison fiable pour un taux probabiliste de faux positifs de 10, 1 et $0,1$ %. Nous n'y avons pas ajouté de la visibilité causale sans le mécanisme car elle est parfois inaténiable sur toute la durée de la simulation.[9] Nous voyons que nous obtenons des latences comparables à une livraison sans ordre (mais sans mécanisme de re-demande). Le 99ème centile des cas 10% et 1% étant systématiquement meilleur que la visibilité sans ordre, et systématiquement inférieur à 22ms. Ceci, contrairement aux figures 4 et 5, où la visibilité causale était systématiquement décalée par rapport à la visibilité réseau.

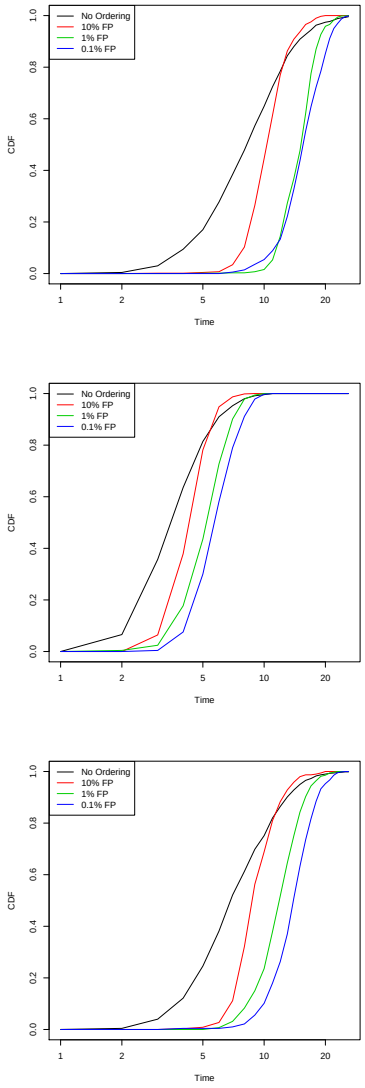

FIGURE 6: Résultat du mécanisme de livraison fiable (dans l'ordre LNKD-SDD, LNKD-HDD, YMMR-W, YMMR-R)

VI. ÉTAT DE L'ART

Plusieurs travaux se sont intéressé à une étude théorique et pratique du comportement de la latence ou de la consistance dans diverse systèmes distribués, mais sans prendre en compte la causalité. Par exemple, [13] s'intéresse au problème de la confirmation TCP en essayant de minimiser de manière probabiliste le surcoût en latence du à la confirmation TCP. [3] s'intéresse au comportement de la consistance inéluctable par quorum partiels et défini une obsolescence bornée probabiliste. [10] et [4] détectent de manière théorique ou pratique les différences dans des systèmes de réplication optimiste. Aussi, des études ont été conduite pour évaluer les performances des algorithmes de réplication utilisant une délivrance causale [1], mais sans étude de la latence.

D'autres travaux proposent des protocoles pour fournir une consistance causale à très grande échelle [2], [17], mais en supposant une communication fiable et sans surcoût.

VII. CONCLUSION ET PERSPECTIVES

Dans cet article, nous nous sommes intéressé aux comportement des systèmes distribués dont les évènements sont soumis à un ordre partiel. Nous avons décrit de manière probabiliste la latence et la visibilité dans ces systèmes. De plus, nous avons montré comment ces définitions permettent de définir un mécanisme de délivrance fiable qui a un surcoût en charge réseau paramétrable et faible.

Dans la suite de nos travaux nous pensons comparer la latence, et la charge réseaux, d'un système utilisant ce mécanisme par rapport aux systèmes de confirmation à paramétrage probabiliste. Nous comptons aussi, de la même

8. Usuellement, seules les écritures ont besoin d'être ordonnées causalement.

9. En effet, il suffit d'un envoi de message ayant une latence élevé (supérieur à la durée de simulation) pour bloquer complètement tous le système.

manière que nous l'avons fait pour la loi exponentielle, instancier nos définitions probabilistes pour des lois de probabilité pareto et mixte exponentielle – pareto.

REMERCIEMENTS

Nous remercions Samir Youcef pour ses commentaires intéressants sur nos travaux. Ces travaux ont été partiellement financé par le projet European FP7 – 609 551, SyncFree (2013–2016).

RÉFÉRENCES

[1] Mehdi Ahmed-Nacer, Claudia-Lavinia Ignat, Gérald Oster, Hyun-Gul Roh, and Pascal Urso. Evaluating crdts for real-time document editing. In ACM, editor, *ACM Symposium on Document Engineering*, page 10 pages, San Francisco, CA, USA, september 2011.

[2] Peter Bailis, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Bolt-on causal consistency. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 761–772, New York, NY, USA, 2013. ACM.

[3] Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica. Probabilistically bounded staleness for practical partial quorums. *Proc. VLDB Endow.*, 5(8) :776–787, April 2012.

[4] David Bermbach and Stefan Tai. Eventual consistency : How soon is eventual ? an evaluation of amazon s3's consistency behavior. In *Proceedings of the 6th Workshop on Middleware for Service Oriented Computing*, MW4SOC '11, pages 1 :1–1 :6, New York, NY, USA, 2011. ACM.

[5] Sebastian Burckhardt, Alexey Gotsman, Hongseok Yang, and Marek Zawirski. Replicated data types : Specification, verification, optimality. *SIGPLAN Not.*, 49(1) :271–284, January 2014.

[6] Michael Burrows. The chubby lock service for loosely-coupled distributed systems. In *OSDI*, pages 335–350. USENIX Association, 2006.

[7] Paul Dourish and Victoria Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, CSCW '92, pages 107–114, New York, NY, USA, 1992. ACM.

[8] Colin J Fidge. Timestamps in message-passing systems that preserve the partial ordering. In *Proc. of the 11th Australian Computer Science Conference (ACSC'88)*, pages 56–66, 1988.

[9] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33 :51–59, June 2002.

[10] Wojciech Golab, Xiaozhou Li, and Mehul A. Shah. Analyzing consistency properties for fun and profit. In *Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '11, pages 197–206, New York, NY, USA, 2011. ACM.

[11] Ning Gu, Qiwei Zhang, Jiangming Yang, and Wei Ye. Dcv : a causality detection approach for large-scale dynamic collaboration environments. In *GROUP '07 : Proceedings of the 2007 international ACM conference on Supporting group work*, pages 157–166, New York, NY, USA, 2007. ACM.

[12] Van Jacobson. Congestion avoidance and control. In *ACM SIGCOMM computer communication review*, volume 18, pages 314–329. ACM, 1988.

[13] Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic tcp acknowledgement and other stories about e/(e-1). In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC '01, pages 502–509, New York, NY, USA, 2001. ACM.

[14] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7) :558–565, 1978.

[15] M. Lanza, L. Hattori, and A. Guzzi. Supporting collaboration awareness with real-time visualization of development activity. In *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on*, pages 202–211, March 2010.

[16] Ka-Cheong Leung and VOK Li. Transmission control protocol (tcp) in wireless networks : issues, approaches, and challenges. *IEEE Communications Surveys &amp ; Tutorials*, 8(4) :64–79, 2006.

[17] Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. Don't settle for eventual : scalable causal consistency for wide-area storage with cops. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 401–416, New York, NY, USA, 2011. ACM.

[18] Stéphane Martin, Mehdi Ahmed-Nacer, and Pascal Urso. Abstract unordered and ordered trees crdt. Rapport de recherche RR-7825, INRIA, December 2011.

[19] Friedemann Mattern. Virtual time and global states of distributed systems. In Michel Cosnard et al., editor, *Proceedings of the International Workshop on Parallel and Distributed Algorithms*, pages 215–226, Château de Bonas, France, October 1989. Elsevier Science Publishers.

[20] Gérald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine. Data Consistency for P2P Collaborative Editing. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW 2006*, pages 259–267, Banff, Alberta, Canada, nov 2006. ACM Press.

[21] Ravi Prakash, Michel Raynal, and Mukesh Singhal. An adaptive causal ordering algorithm suited to mobile computing environments. *J. Parallel Distrib. Comput.*, 41(2) :190–204, 1997.

[22] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In Xavier Défago, Franck Petit, and V. Villain, editors, *Stabilization, Safety, and Security of Distributed Systems (SSS)*, volume 6976, pages 386–400, Grenoble, France, October 2011.

[23] Chengzheng Sun and Clarence A. Ellis. Operational transformation in real-time group editors : Issues, algorithms, and achievements. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work - CSCW'98*, pages 59–68, New York, New York, États-Unis, November 1998. ACM Press.

[24] Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 5(1) :63–108, March 1998.

[25] R Core Team. R language definition, 2000.

[26] Werner Vogels. Eventually consistent. *Commun. ACM*, 52(1) :40–44, January 2009.

[27] S. Weiss, P. Urso, and P. Molli. Logoot : A scalable optimistic replication algorithm for collaborative editing on p2p networks. In *29th IEEE International Conference on Distributed Computing Systems (ICDCS 2009)*, pages 404 –412, Montréal, Québec, Canada, jun. 2009. IEEE Computer Society.